

EE 610, ST: ML Fundamentals

Dataset Partitioning and Performance Evaluation Metrics

Dr. W. D. Pan

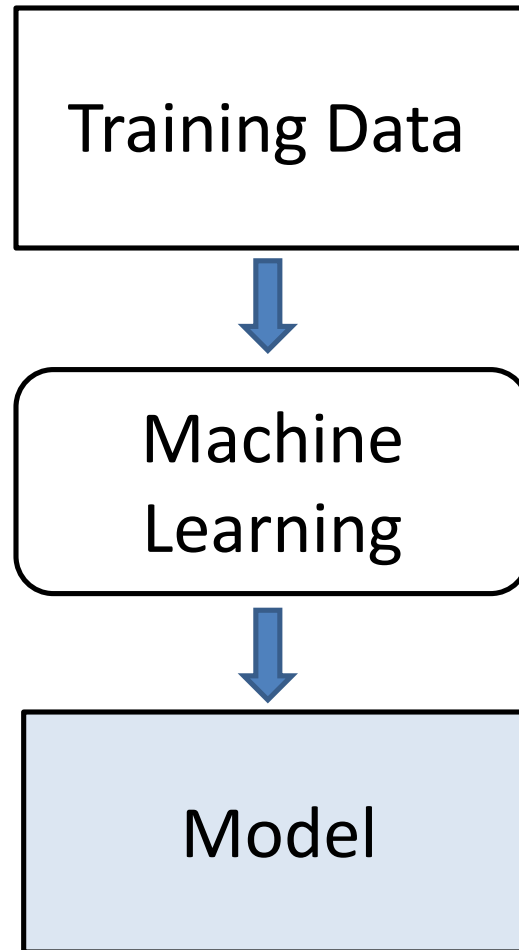
Dept. of ECE

Univ. of Alabama in Huntsville

Topics

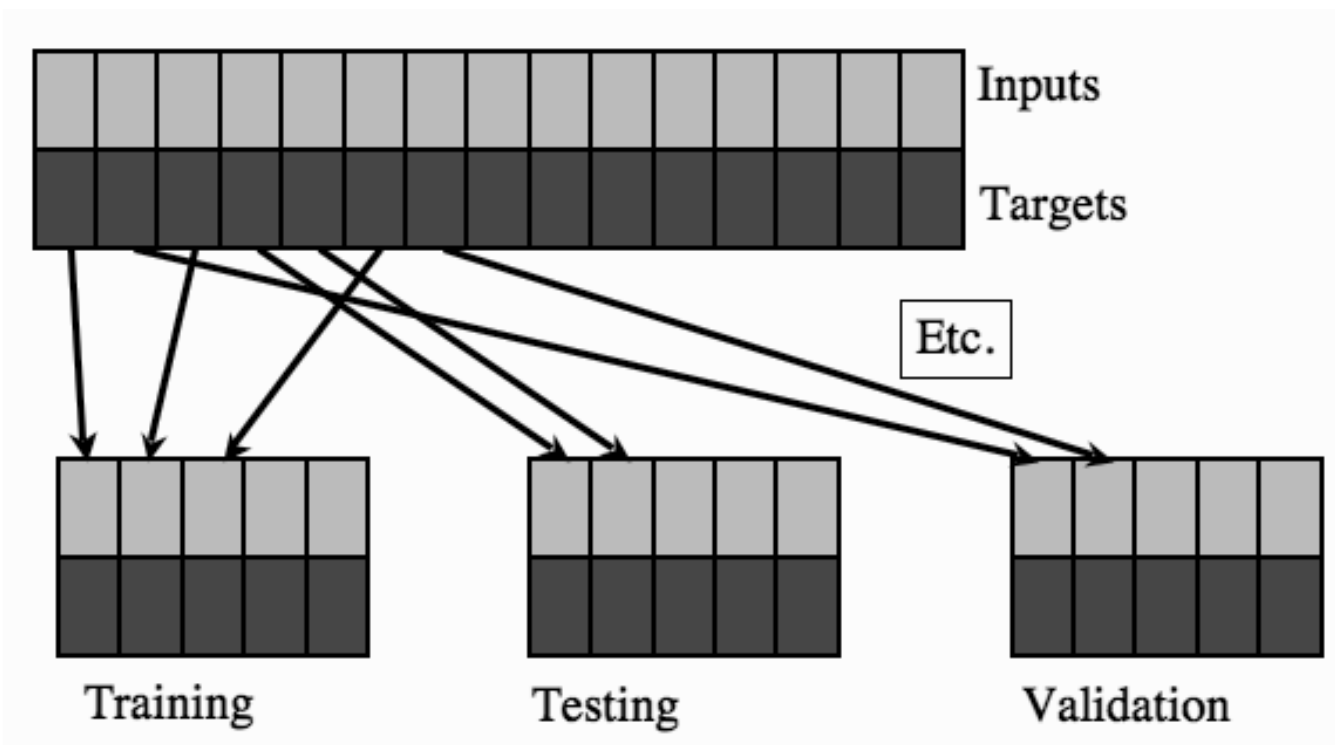
- Dataset Partitioning Methods
- Cross Validation
- Hold out
- Resubstitution
- Stratification Sampling
- Performance Evaluation Metrics
- Confusion Matrix
- ROC

ML Process



- Training
 - Given the dataset, algorithm, and parameters, training is the use of computational resources in order to build a model of the data in order to predict the outputs on new data.
- Evaluation (testing)
 - Before a system can be deployed it needs to be tested and evaluated for accuracy on data that it was not trained on.
 - This can often include a comparison with human experts in the field, and the selection of appropriate metrics for this comparison.

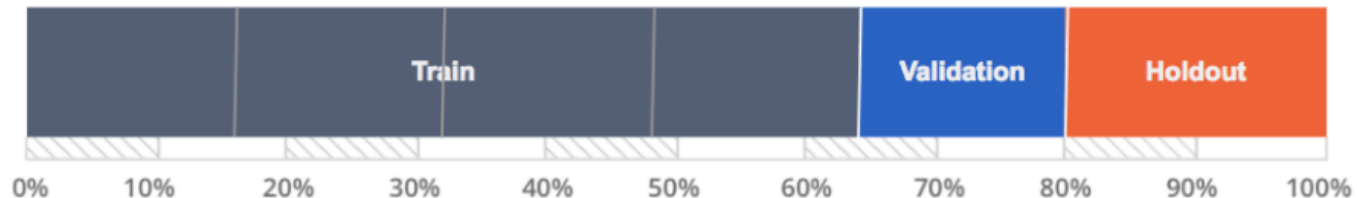
Training, Testing, and Validation Sets

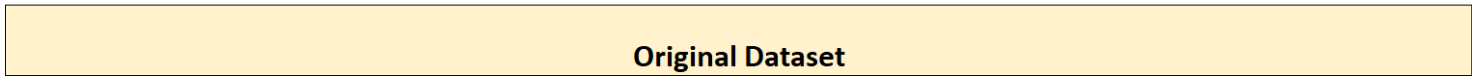


- For large datasets, sometimes only training and testing partition;
- For small datasets, cross-validation during the training phase: training (training/validation) and testing partition

Dataset Partitioning Methods

- Training and Testing Partition
 - Partitions data randomly into exactly two subsets of specified ratio for **training**, and **testing (holdout)**.
 - This method performs training and testing only once, which cuts computation time on large datasets.
 - We should interpret the reported prediction errors with caution on small data sets.
- Training/Validation and Testing Partition
 - The dataset is split into different groups, some for **training**, some for **validation**, and the rest for **testing**.
 - The **validation** set is the holdout during the training phase, and is used to optimize the model parameters.
 - The **testing** set is used to provide an unbiased estimate of the final model after cross-validation.





Split into Training Set and Test Set



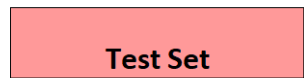
Perform k-fold cross validation



Produce final model



Use test set to get unbiased estimate of final model performance



Need for Cross Validation

- Cross-validation is a model assessment technique, based on a resampling procedure.
- Cross-validation is used to evaluate a machine learning algorithm's performance in making predictions on new datasets that it has not been trained on.
- When training a model with a small dataset, there is the risk of either *overfitting* or *underfitting* a model.
- The choice of training set and testing set are critical in reducing the above risk.
- This is why cross-validation comes into practice.

K-Fold Cross Validation

- The procedure has a single parameter called K , which is the number of subsets (groups) that a given data point (or sample) is to be split into.
- This approach involves randomly dividing the set of observations into K groups, or folds, of approximately equal size.
- As such, the procedure is often called K -fold cross-validation. For example, $K = 10 \Rightarrow$ 10-fold cross-validation.
- Cross validation generally results in a less biased or less optimistic estimate of the model **skill (predictive performance)** than other methods, such as a simple train/test split.

The Procedure

- Shuffle the dataset randomly
- Split the dataset into K groups
- For each unique group:
 - Take the group as a holdout or validation dataset
 - Use the remaining groups as a training data set
 - Fit a model on the training set and evaluate it on the validation set (holdout)
 - Retain the evaluation score
 - Summarize the skill of the model based on the resulting K evaluation scores

Leave-some-out, K-fold Cross-Validation

4-fold validation (k=4)



Note: “Testing set” here actually means “validation set”, to distinguish it from the “testing set” (hold out) set aside to test the final model obtained after cross-validation.

In the literature, “testing” is often used loosely in place of validation.

Model Skill Scores

- Cross-validation is primarily used in applied machine learning to estimate the **skill** of a machine learning model on unseen data.
- We can use a small dataset to estimate how the trained model is expected to perform in general, when the model is used to make predictions on data not used during the training of the model.
- The results of a K-fold cross-validation runs are often summarized with the mean of the model skill scores.
- It is also good practice to include a measure of the variance of the skill scores, such as the standard deviation.

Summary

- Cross-validation can be a computationally intensive operation, since training and validation is done multiple times.
- Because each partition group is independent, we can perform this analysis in parallel to speed up the process.
- For larger datasets, techniques like holdout or resubstitution are recommended
- For small datasets, we can use K-fold cross validation, or repeated random sub-sampling methods, etc.
 - Repeated random sub-sampling: Creating multiple random partitions of data to use as training set and testing set using the Monte Carlo methodology and aggregates results over all the runs.
 - This technique has a similar idea to the K-fold method, but each test set is chosen independently, which means some data points might be used for testing more than once.

Special Cases

- Resubstitution validation
 - Does not partition the data and all data is used for training the model.
 - Resubstitution error
 - The error rate is evaluated based on the prediction outcome using the trained model, versus the actual value from the **same** training data set.
- This approach often produces overly optimistic estimates for performance and should be avoided if there is sufficient data.

Example:

Classification loss for naive Bayes classifiers by resubstitution

```
>> L = resubLoss(Mdl)
```

- returns the in-sample minimum misclassification cost loss (L), which is a scalar representing how well the trained naive Bayes classifier Mdl classifies the predictor data stored in Mdl.X as compared to the true class labels stored in Mdl.Y.

Dataset Content

https://en.wikipedia.org/wiki/Iris_flower_data_set#Data_set

```
>> load fisheriris
```

```
    meas          150x4 double
    species 150x1 cell
```

```
>> meas(1,:)
```

```
ans =
```

```
5.1000  3.5000  1.4000  0.2000
```

```
>> species(1)
```

```
ans =
```

```
1x1 cell
```

```
array
```

```
    {'setosa'}
```

```
>>
```

```
species(51)
```

```
ans =
```

```
1x1 cell
```

```
array
```

```
    {'versicolor'}
```

```
>>
```

```
species(101)
```

```
ans =
```

```
1x1 cell
```

```
array
```

```
    {'virginica'}
```

Matlab (Statistical and Machine Learning Toolbox)

```
>> load fisheriris
>> X = meas;
>> Y = species;
>> Mdl = fitcnb(X,Y)    % Train a naïve Bayes classifier
Mdl =
  ClassificationNaiveBayes
      ResponseName: 'Y'
      ...
  ClassNames: {'setosa' 'versicolor' 'virginica'}
  NumObservations: 150
  DistributionNames: {'normal' 'normal' 'normal' 'normal'}

>> L = resubLoss(Mdl)
L =
  0.0400
```

The naive Bayes classifier misclassifies 4% of the training observations.

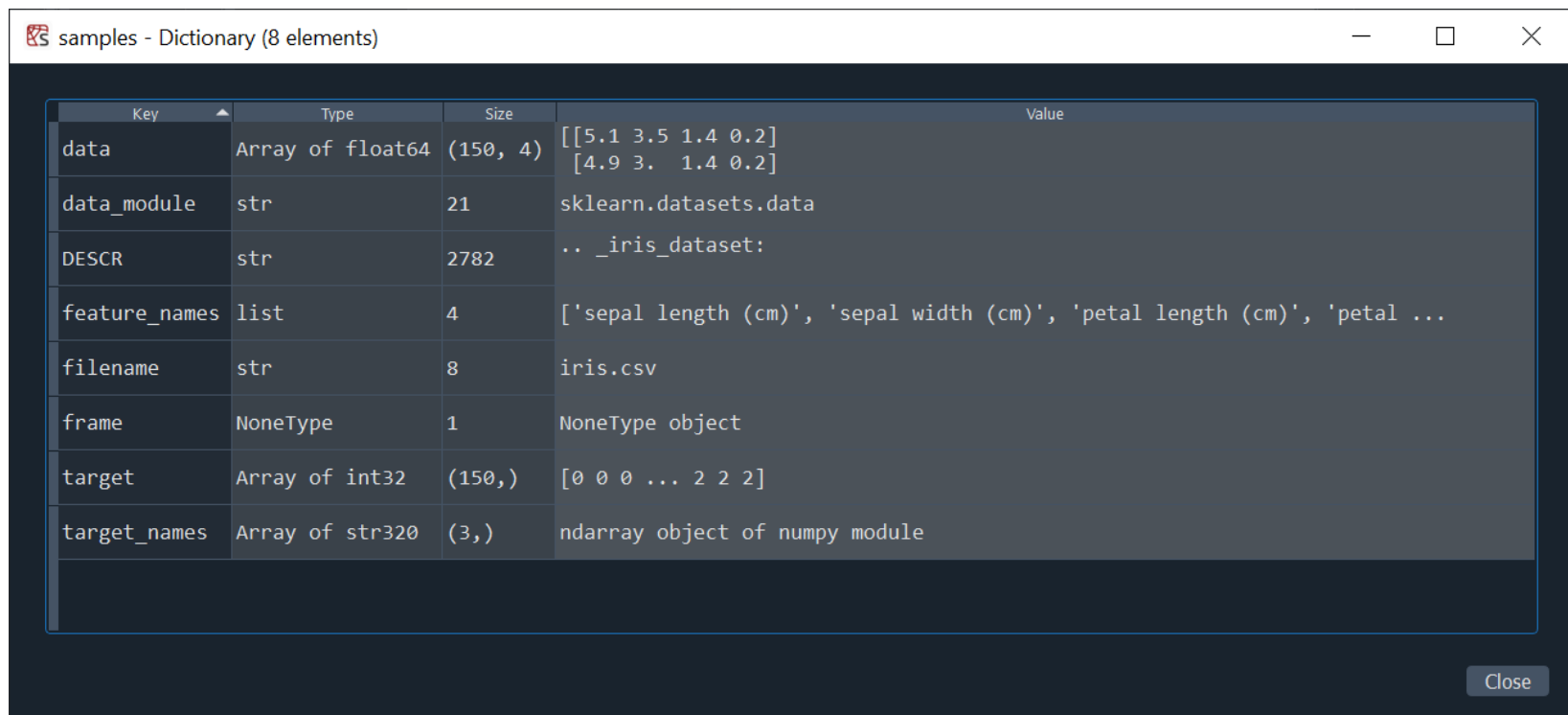
Prediction (Resubstitution)

```
>> X = meas;
>> Y = species;
>> Mdl = fitcnb (X, Y);           % Train the classifier with (X, Y)

>> label = predict (Mdl, X);     % Predict using the same training data X
>> diff = strcmp (label, Y);
>> length(find(diff ==0))        % Number of prediction errors
ans =
     6
>> 6/150
ans =
    0.0400
```

Fisher Iris Dataset

```
>>> from sklearn.datasets import load_iris
>>> samples = load_iris()
>>> samples.target_names
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```



samples - Dictionary (8 elements)

Key	Type	Size	Value
data	Array of float64	(150, 4)	[[5.1 3.5 1.4 0.2] [4.9 3. 1.4 0.2]
data_module	str	21	sklearn.datasets.data
DESCR	str	2782	.. _iris_dataset:
feature_names	list	4	['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal ...
filename	str	8	iris.csv
frame	NoneType	1	NoneType object
target	Array of int32	(150,)	[0 0 0 ... 2 2 2]
target_names	Array of str320	(3,)	ndarray object of numpy module

Close

Naïve Bayes Classifier in sklearn

```
from sklearn.naive_bayes import GaussianNB  
clf = GaussianNB()
```

```
from sklearn.datasets import load_iris  
train_samples = load_iris()
```

```
X = train_samples.data  
Y = train_samples.target
```

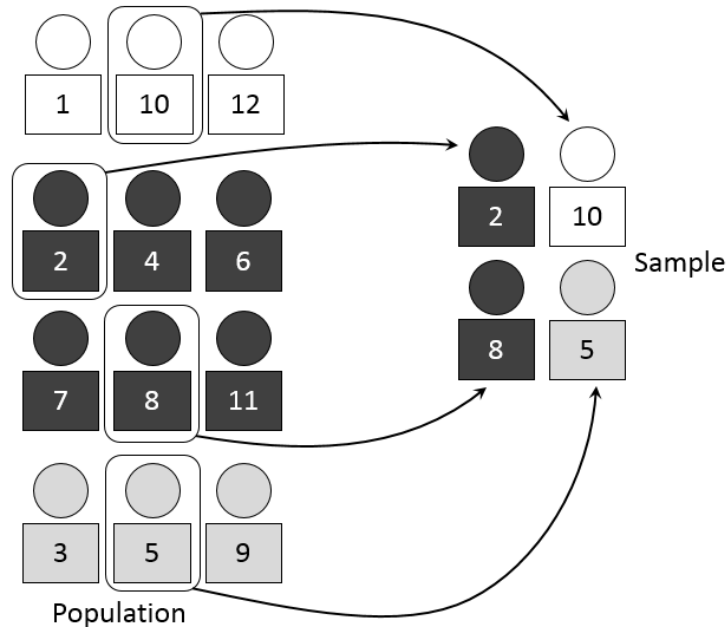
```
clf.fit (X, Y)  
label = clf.predict (X)
```

```
import numpy as np  
np.sum (label != Y)           # Out: 6  
Y.size                        # Out: 150  
np.sum (label != Y)/Y.size   # Out: 0.04
```

Stratification Sampling

- Stratified sampling is a sampling technique where the samples are selected in the same proportion as they appear in the population, by dividing the population into groups called 'strata' based on a characteristic.
- Implementing stratified sampling ensures the training and testing sets, or training and validation sets have the same proportion of the feature of interest as in the original dataset.
- Thus the final trained model has a error performance that is a close approximation to the generalization error.

Example of Stratification



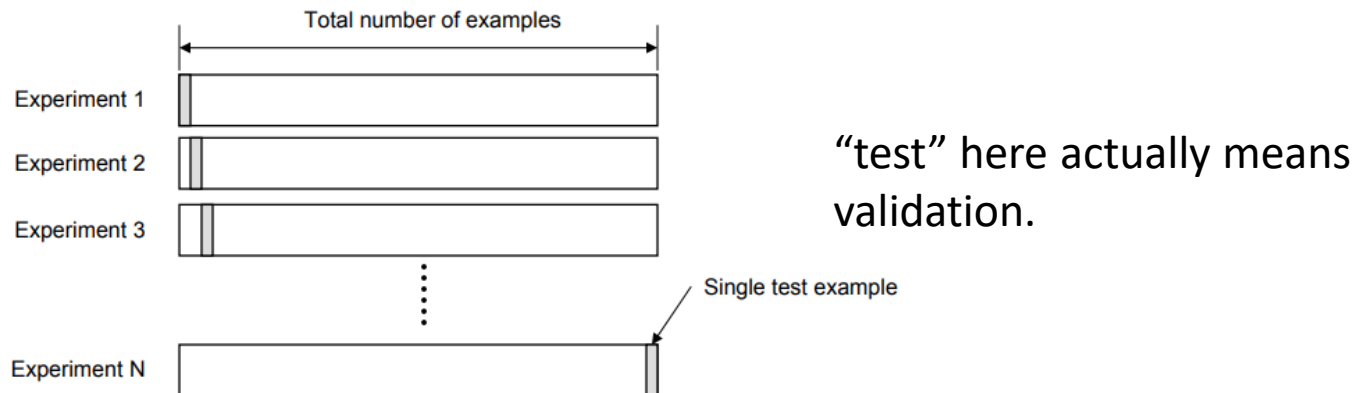
```
>> c = cvpartition (group, 'KFold', k)
```

- creates a random partition for **stratified** k-fold cross-validation.
- Each subsample, or fold, has approximately the same number of observations and contains approximately the same class proportions as in **group**.

LOOCV

Leave-one-out cross-validation

- Cross validation is performed by partitioning the dataset, and leaving out only one data point to validate the trained model, and using the remaining data points for training the model.



`c = cvpartition(n, 'Leaveout')`

- creates a random partition for leave-one-out cross-validation on n observations.
- Leave-one-out is a special case of 'KFold' in which the number of folds equals the number of observations.

'cvpartition_demo.m'

```
N = 15;  
X1 = -3+randn(1,N);  
C1 = ones(1,N);    % labels
```

```
X2 = 3+randn(1,N);  
C2 = 2*ones(1,N);
```

```
X = [X1 X2];  
C = [C1 C2];
```

```
cv = cvpartition(C, 'KFold', 5)  
C(cv.test(1))  
C(cv.test(2))
```

```
...  
cv = cvpartition(C, 'KFold', 5, 'Stratify', false)  
cv = cvpartition(C, 'HoldOut', 0.2)  
cv = cvpartition(C, 'HoldOut', 0.2, 'Stratify', false)
```

```
% Generate smaller dataset  
cv = cvpartition(C, 'LeaveOut')
```

```
cv = cvpartition(C,'resubstitution')
```

```
cv =  
K-fold cross validation partition  
  NumObservations: 30  
  NumTestSets: 5  
  TrainSize: 24 24 24 24 24  
  TestSize: 6 6 6 6 6
```

```
C(cv.test(1))  
ans =  
    1    1    1    2    2    2
```

```
C(cv.test(2))  
ans =  
    1    1    1    2    2    2
```

'cross_validate_demo.py'

```
from sklearn.model_selection import StratifiedKFold
```

```
cv = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 1)
```

```
from sklearn.model_selection import KFold
```

```
cv = KFold(n_splits = 5, shuffle = True, random_state = 1)
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, C_train, C_test = train_test_split \  
    (X, C, test_size = 0.2, random_state=1)
```

```
from sklearn.model_selection import LeaveOneOut
```

```
cv = LeaveOneOut ()
```


Cross Validation Score Report

- The results of a K-fold cross-validation runs are often summarized with the mean of the model skill scores.
- It is also good practice to include a measure of the variance of the skill scores, such as the standard deviation.
- Matlab
 - kfoldloss function
- Sklearn
 - cross_val_score

'kfoldloss_demo.m'

```
K = 10          % K-fold cross validation
load fisheriris

rng(1);        % For reproducibility

Model_cv = fitcnb(meas, ...
species, 'CrossVal', 'on', 'KFold', K);
Model_cv;
Model_cv.Partition;

error_rate_1 =
kfoldLoss(Model_cv, 'LossFun', 'ClassifErr')
```

```
% Alternatively, use cvpartition function
rng(1); % For reproducibility
cv_part = cvpartition (species, 'KFold', K);

Errors = zeros(1, K);

for i = 1: K
    training_index = cv_part.training(i);
    testing_index = cv_part.test(i);

    Model = fitcnb(meas(training_index,:), species(training_index,:));

    % Then predict using the testing data
    label = predict(Model, meas(testing_index,:));
    diff = strcmp (label, species(testing_index,:));
    Errors(i) = length(find(diff==0))/length(label);

end

error_rate_2 = mean(Errors)
```

sklearn

```
""" 'cross_val_score.py' """
```

```
from sklearn.naive_bayes import GaussianNB  
clf = GaussianNB()
```

```
from sklearn.datasets import load_iris  
train_samples = load_iris()  
X = train_samples.data  
Y = train_samples.target
```

```
from sklearn.model_selection import StratifiedKFold  
KFold1 = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
```

```
from sklearn.model_selection import cross_val_score
```

```
scores1 = cross_val_score(clf, X, Y, cv = KFold1, scoring = 'accuracy')  
import numpy as np  
print(np.mean(scores1))  
print(np.std(scores1))
```

Metrics to Evaluate the Quality of Prediction

- TP, TN, FP, FN
- Sensitivity (Recall)
- Specificity
- Precision
- Accuracy
- F1 score
- Confusion Matrix
- Receiver Operator Characteristic (ROC) curve

Binary Classification

Individual Number	1	2	3	4	5	6	7	8	9	10	11	12
Actual Classification	1	1	1	1	1	1	1	1	0	0	0	0
Predicted Classification	0	0	1	1	1	1	1	1	1	0	0	0
Result	FN	FN	TP	TP	TP	TP	TP	TP	FP	TN	TN	TN

		Predicted condition	
		Positive (PP)	Negative (PN)
Actual condition	Total population = P + N	Positive (PP)	Negative (PN)
	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Terminology

- True Positive (TP)
 - A test result that **correctly** indicates the presence of a condition or characteristic
- True Negative (TN)
 - A test result that **correctly** indicates the absence of a condition or characteristic
- False Positive (FP)
 - A test result which **wrongly** indicates that a particular condition or attribute is present
- False Negative (FN)
 - A test result which **wrongly** indicates that a particular condition or attribute is absent

- Accuracy (ACC)

$$\frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN}$$

True positive (TP)	False negative (FN)
False positive (FP)	True negative (TN)

- Sensitivity (**Recall**, True Positive Rate, **TPR**, or Hit Rate)

$$\frac{TP}{P} = \frac{TP}{TP + FN}$$

Positive (P)	True positive (TP)	False negative (FN)
--------------	--------------------	---------------------

- Specificity (Selectivity, or True Negative Rate, **TNR**)

$$\frac{TN}{N} = \frac{TN}{TN + FP}$$

Negative (N)	False positive (FP)	True negative (TN)
--------------	---------------------	--------------------

- Precision (Positive Predictive Value, PPV)

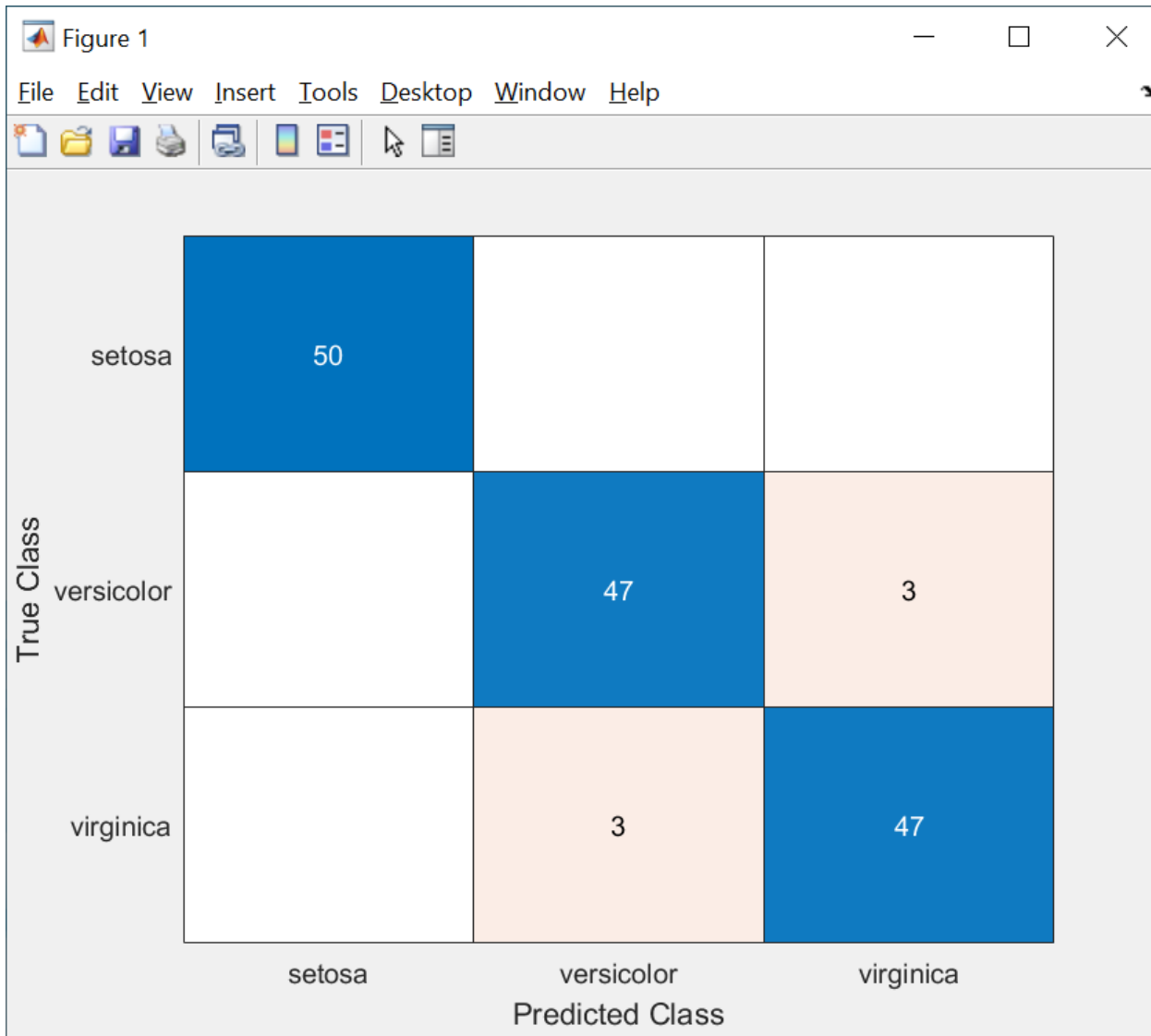
$$\frac{TP}{TP + FP}$$

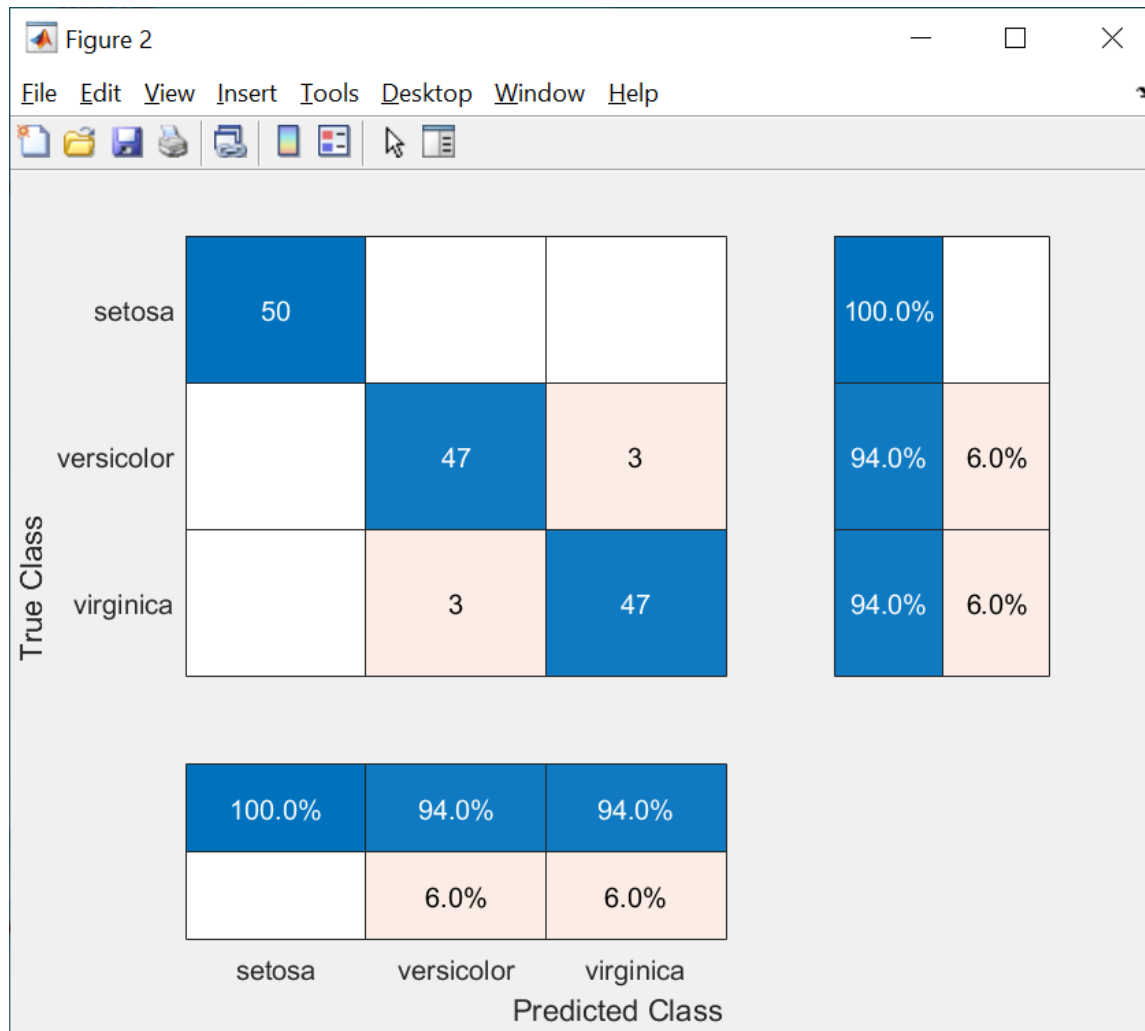
Positive (PP)
True positive (TP)
False positive (FP)

Confusion Matrix (Chart)

- A confusion matrix contains information about known class labels and predicted class labels.
- The (i,j) element in the confusion matrix is the number of samples whose known class label is class i and whose predicted class is j.
- The diagonal elements represent correctly classified observations.

```
>> label = predict (Mdl, X);  
>> Pred_Class = resubPredict (Mdl);  
>> isequal (label, Pred_Class)  
ans =  
    logical  
     1  
>> confusionchart (Y, Pred_Class)           % Y has the true class labels
```





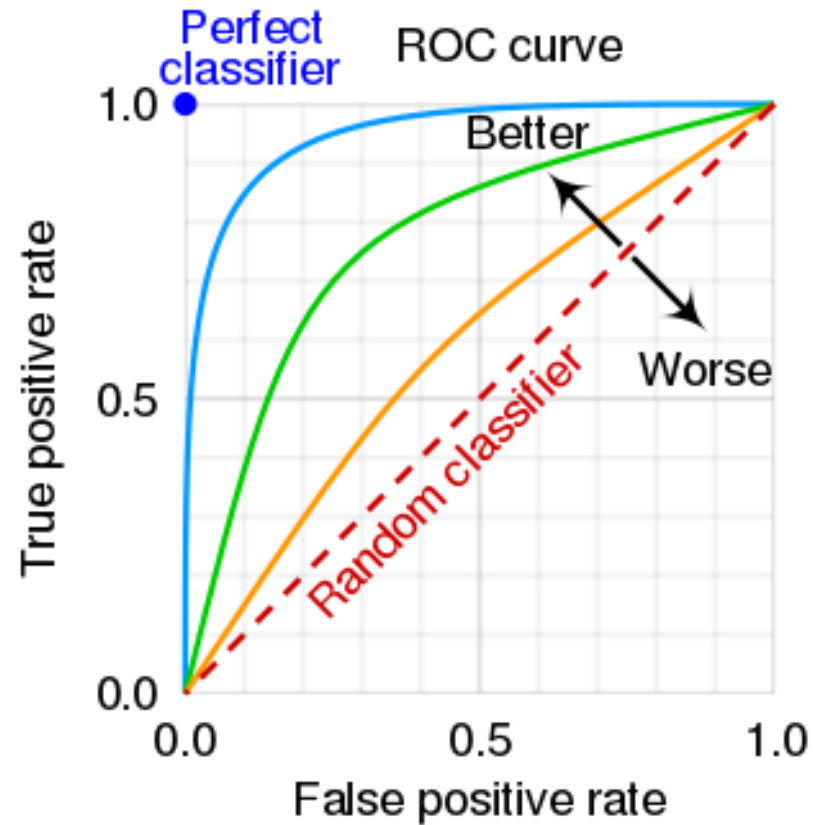
```
>> confusionchart(Y, Pred_Class, ...
    'ColumnSummary','column-normalized', 'RowSummary','row-normalized');
```

The ROC (Receiver Operating Characteristic) Curve

- The metrics discussed previously not only be used to evaluate a particular classifier, but also compare classifiers,
 - either the same classifier with different learning parameters,
 - or completely different classifiers.
- For comparison of different classifiers, the Receiver Operating Characteristic (ROC) curve is useful.
- The ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.
- The ROC curve is created by plotting the **True Positive Rate** (TPR, Sensitivity, Recall) against the **False Positive Rate** (FPR, or **False Alarm Rate**) at various threshold settings.
- The method of ROC analysis was originally developed for operators of military radar receivers to detect objects in battlefields.

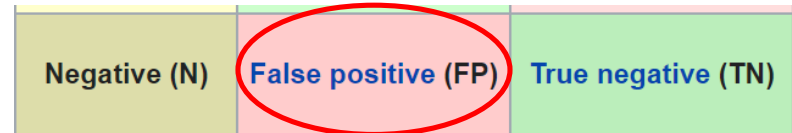
$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

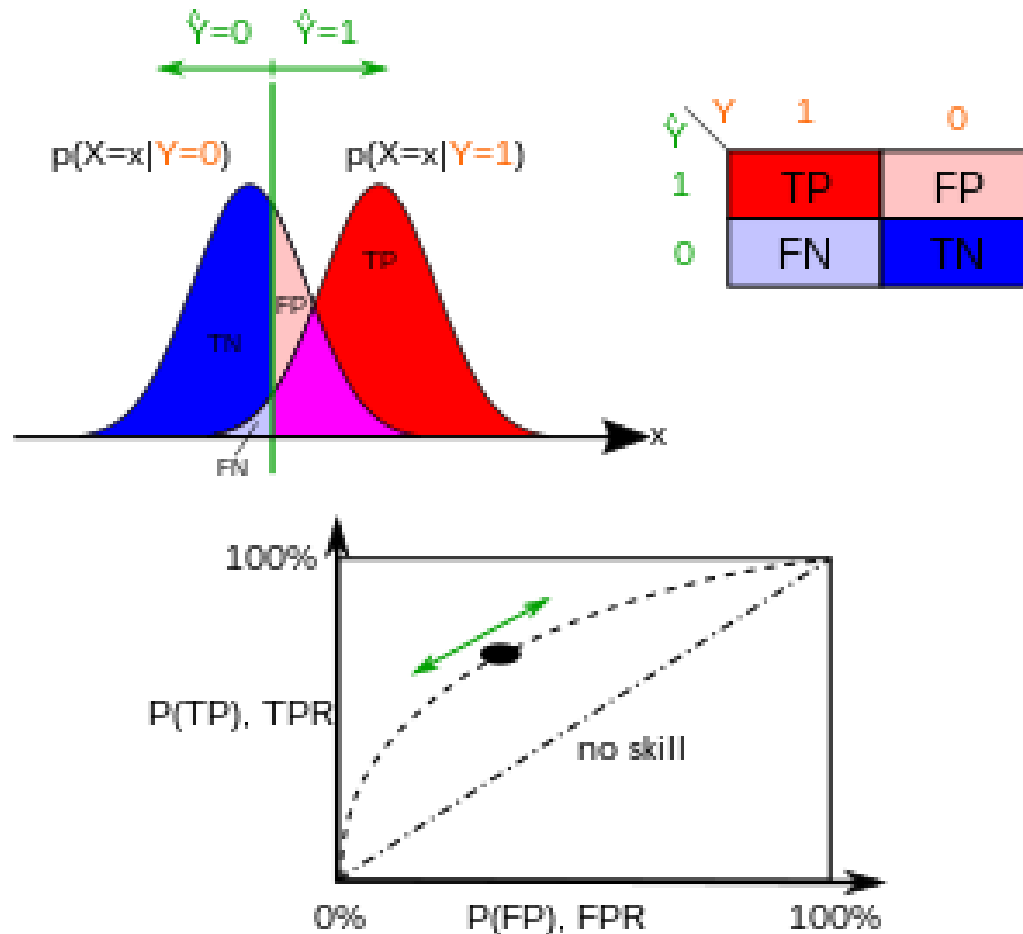


In general, Sensitivity increases with False Alarm Rate.

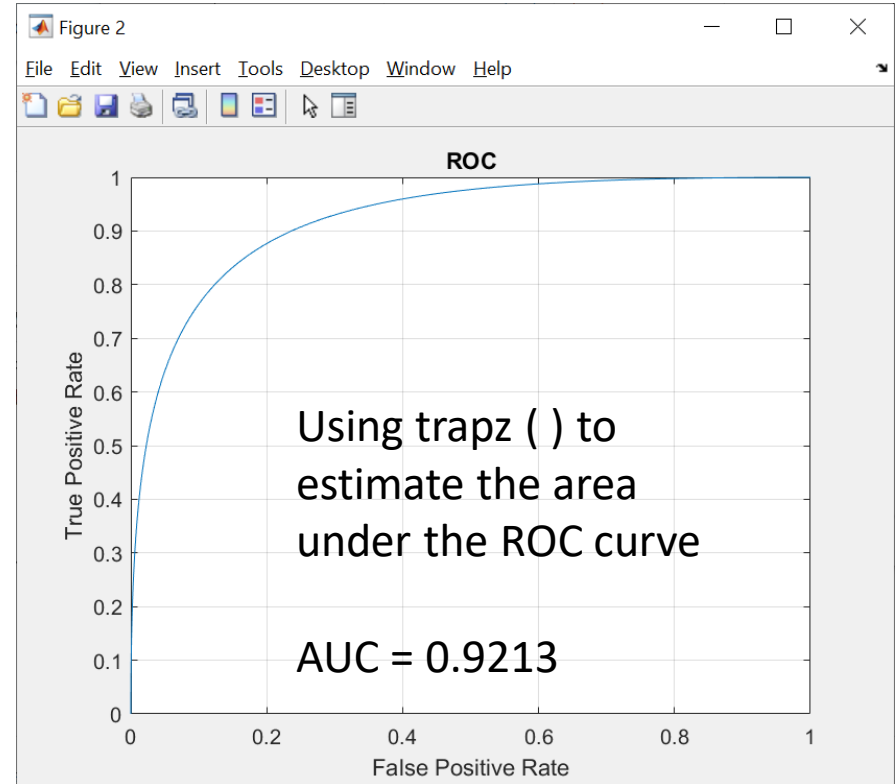
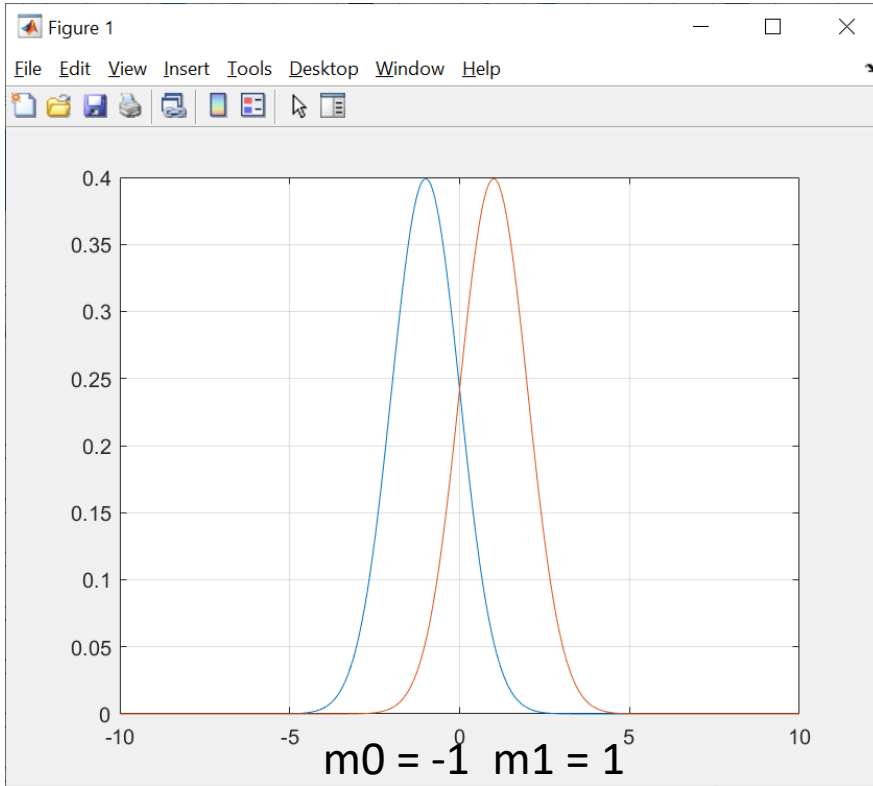
$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$



Binary Classification



roc_pdf.m

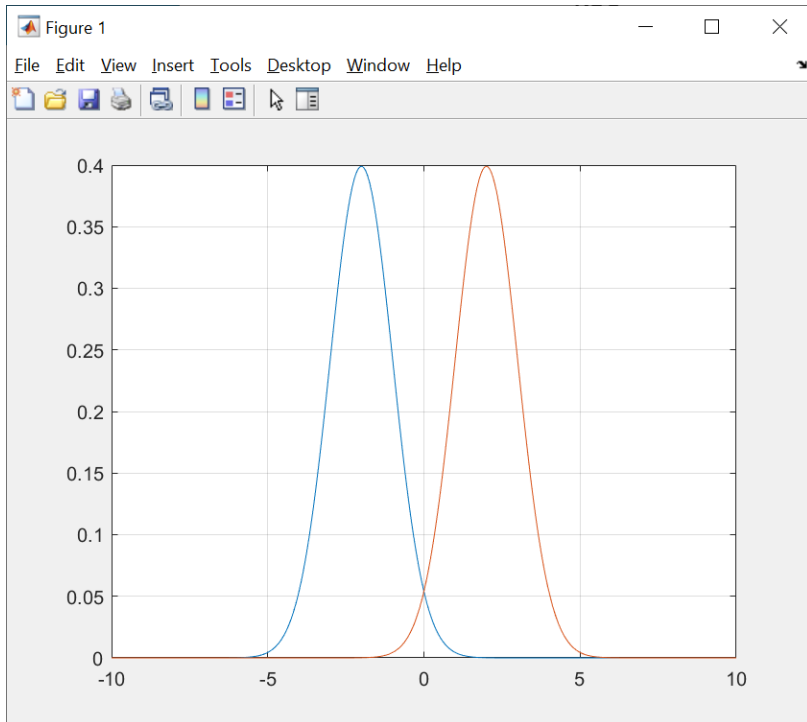


Decision Boundary at $x = 0$

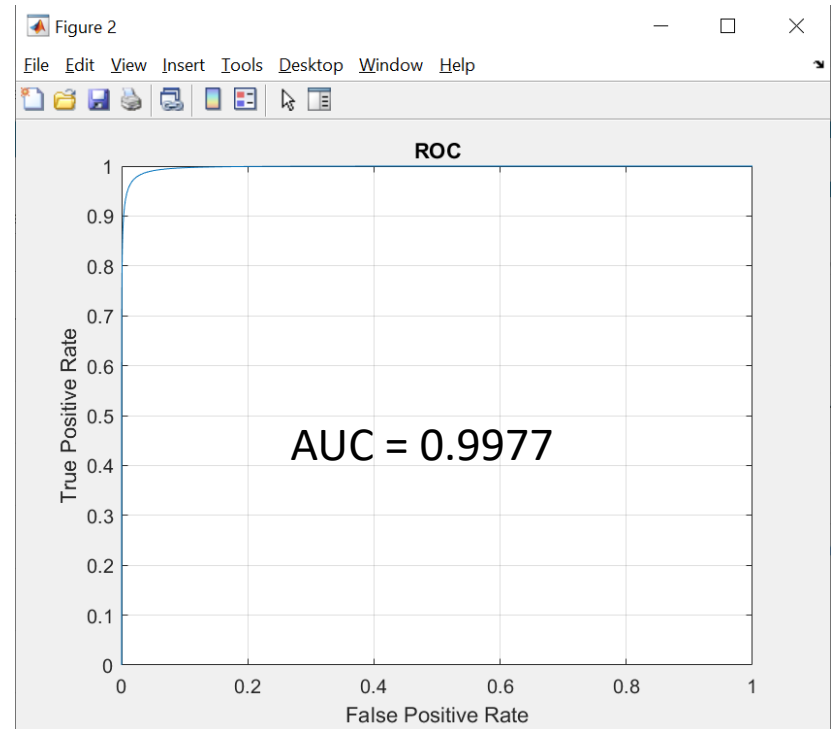
```
>> TPR      >> FPR
TPR =      FPR =
    0.8413    0.1587
```

```
>> % TPR for Class 1
TPR = qfunc((B - m1)/sigma);
% FPR
FPR = qfunc((B - m0)/sigma);
```

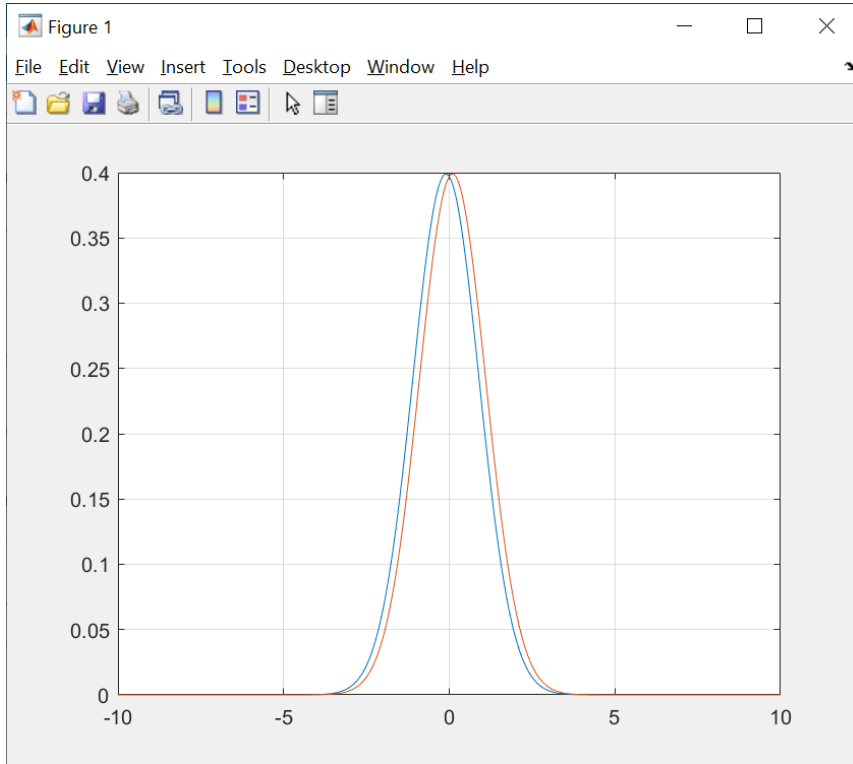
Better Separation



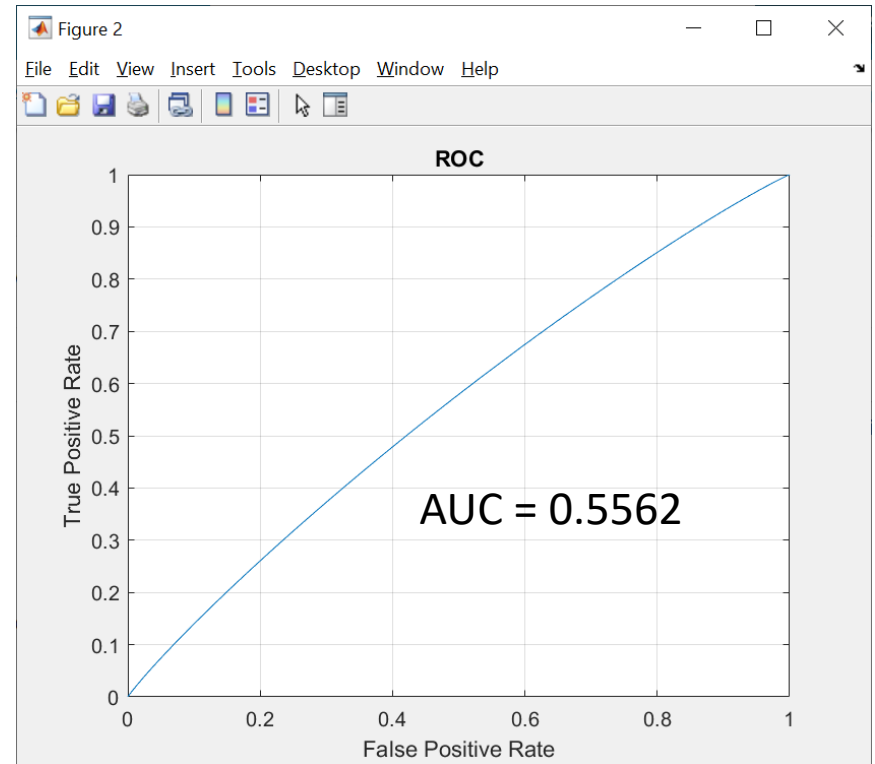
$m_0 = -2$ $m_1 = 2$



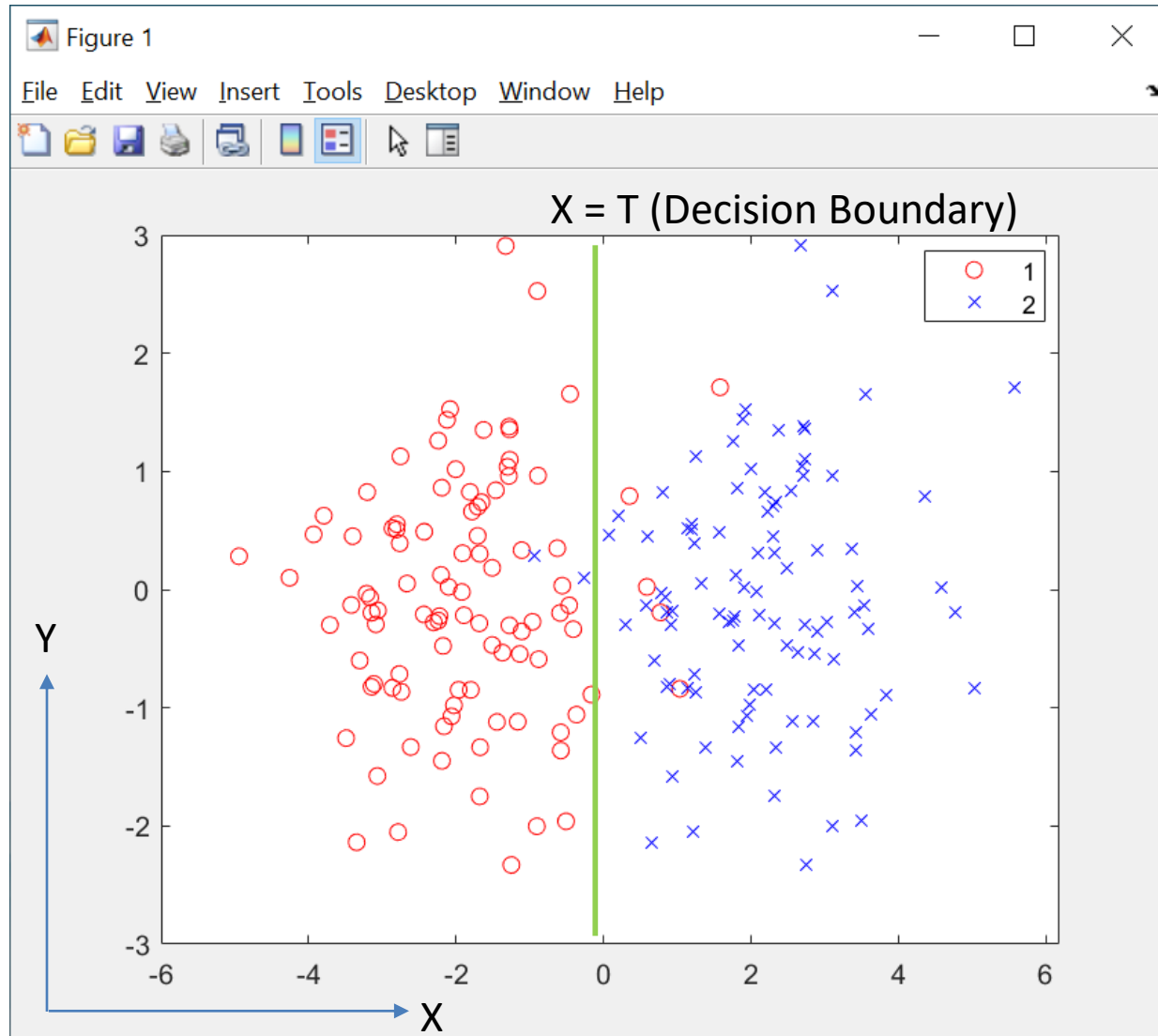
Hard to Separate



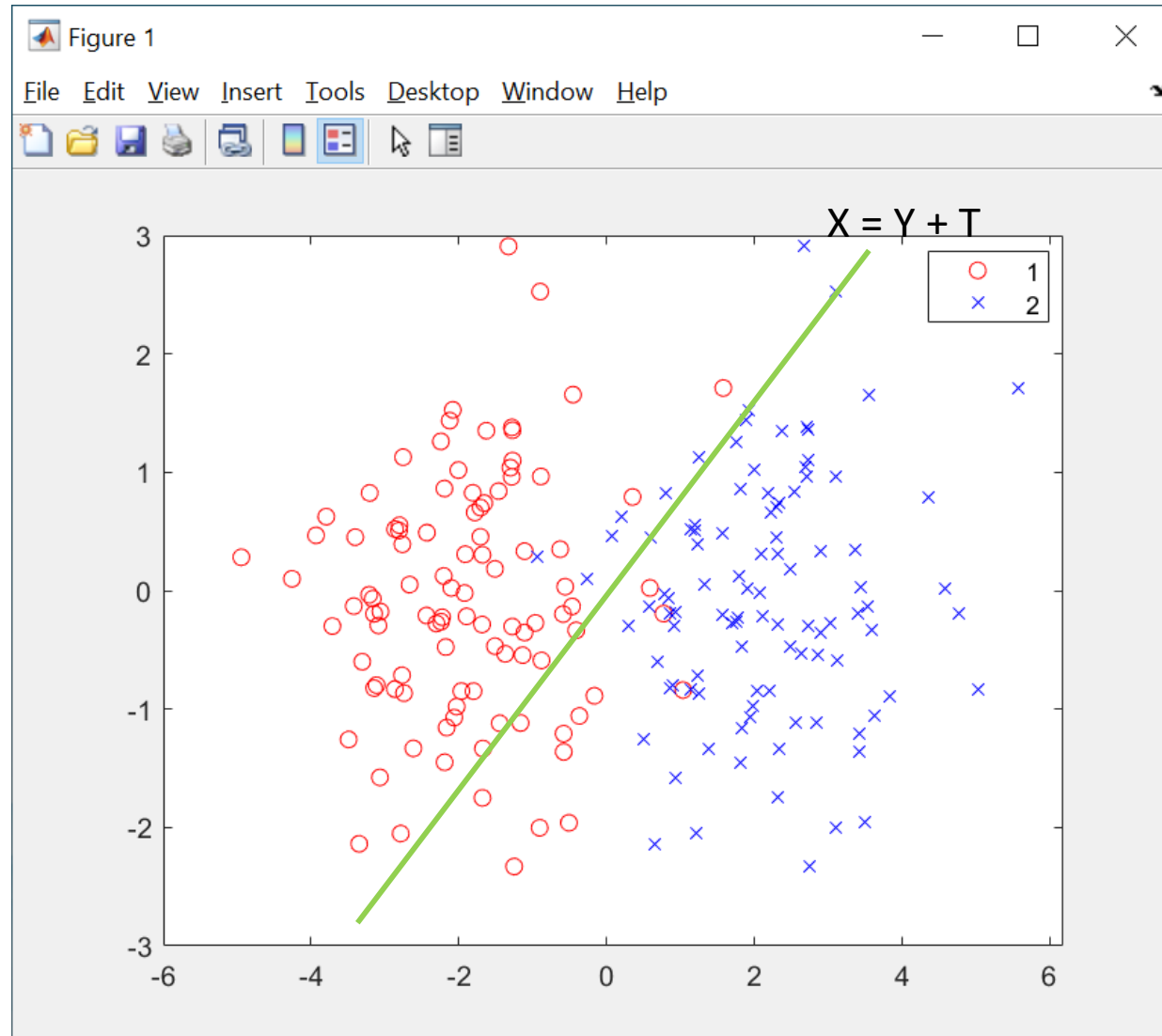
$m_0 = -0.1$ $m_1 = 0.1$



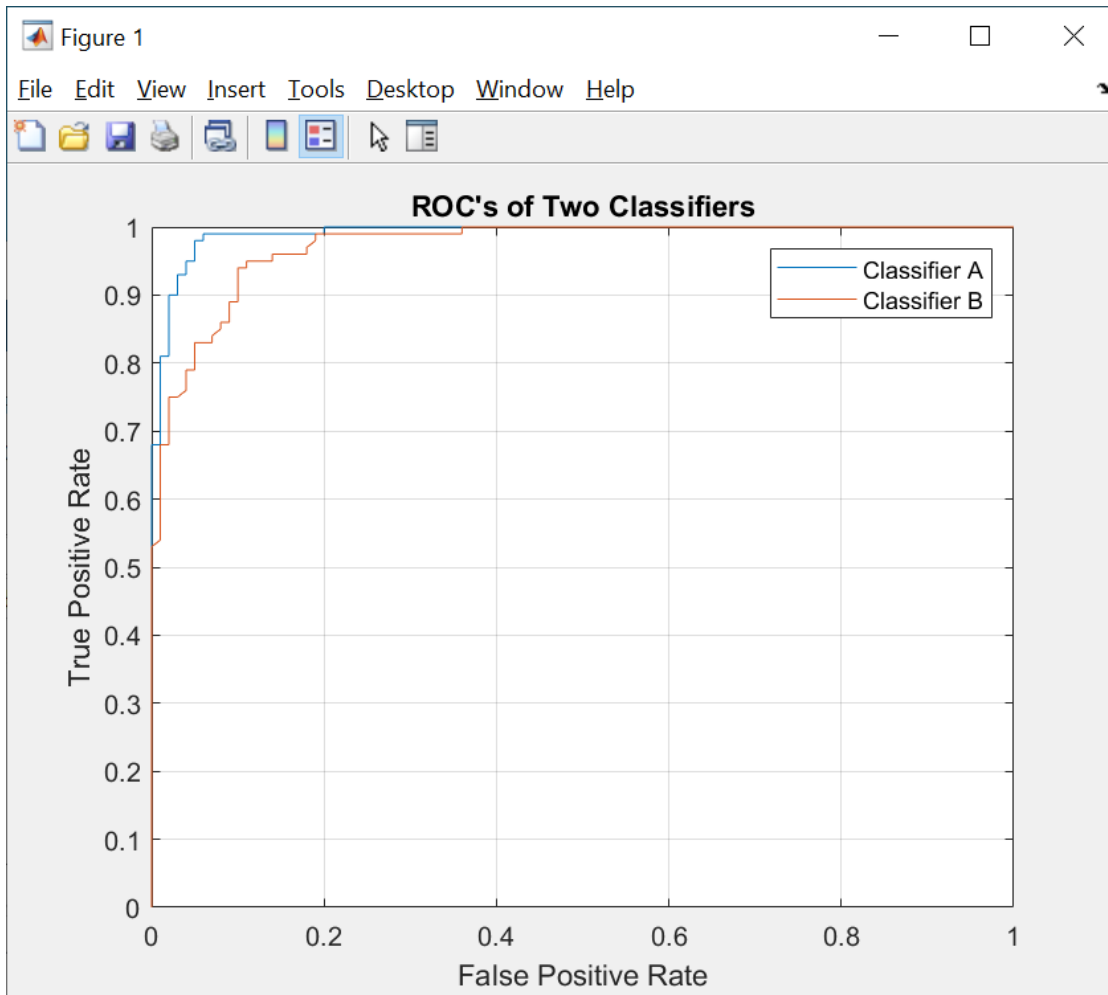
Classifier A



Classifier B



Comparison of ROC's



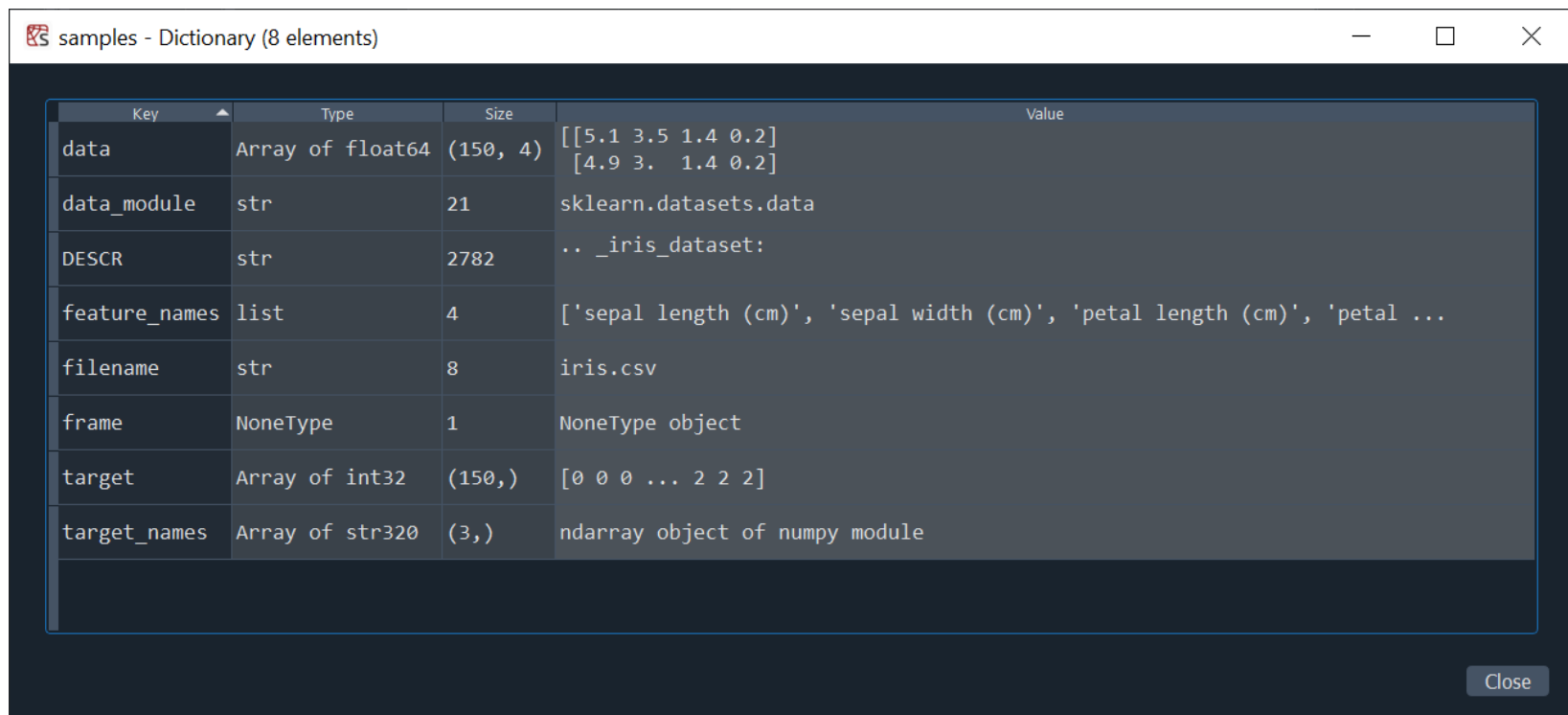
```
>> roc_compare  
AUC_A =  
    0.9911  
AUC_B =  
    0.9720
```

scikit-learn (sklearn): ML in Python

- Python IDE Installation
 - Anaconda Data Science Platform:
<https://www.anaconda.com>
 - Installation includes Python, NumPy, and many other commonly used packages for scientific computing and data science.
 - Includes the Spyder (with IPython, or interactive Python) development environment that supports advanced editing, analysis, debugging, etc.
- scikit-learn is an open source machine learning library that supports supervised and unsupervised learning.
 - <https://scikit-learn.org/stable/index.html>

Fisher Iris Dataset

```
>>> from sklearn.datasets import load_iris
>>> samples = load_iris()
>>> samples.target_names
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```



samples - Dictionary (8 elements)

Key	Type	Size	Value
data	Array of float64	(150, 4)	[[5.1 3.5 1.4 0.2] [4.9 3. 1.4 0.2]
data_module	str	21	sklearn.datasets.data
DESCR	str	2782	.. _iris_dataset:
feature_names	list	4	['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal ...
filename	str	8	iris.csv
frame	NoneType	1	NoneType object
target	Array of int32	(150,)	[0 0 0 ... 2 2 2]
target_names	Array of str320	(3,)	ndarray object of numpy module

Close

Naïve Bayes Classifier in sklearn

```
from sklearn.naive_bayes import GaussianNB  
clf = GaussianNB()
```

```
from sklearn.datasets import load_iris  
train_samples = load_iris()
```

```
X = train_samples.data  
Y = train_samples.target
```

```
clf.fit (X, Y)  
label = clf.predict (X)
```

```
import numpy as np  
np.sum (label != Y)           # Out: 6  
Y.size                       # Out: 150  
np.sum (label != Y)/Y.size   # Out: 0.04
```

Confusion Matrix in sklearn

```
>>> from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
>>> confusion_matrix(Y, label)
array([[50, 0, 0],
       [ 0, 47, 3],
       [ 0, 3, 47]], dtype=int64)
>>> cm = confusion_matrix(Y, label)
>>> disp = ConfusionMatrixDisplay
...     (confusion_matrix = cm,
...     display_labels = train_samples.target_names)
```


Visualization in Python

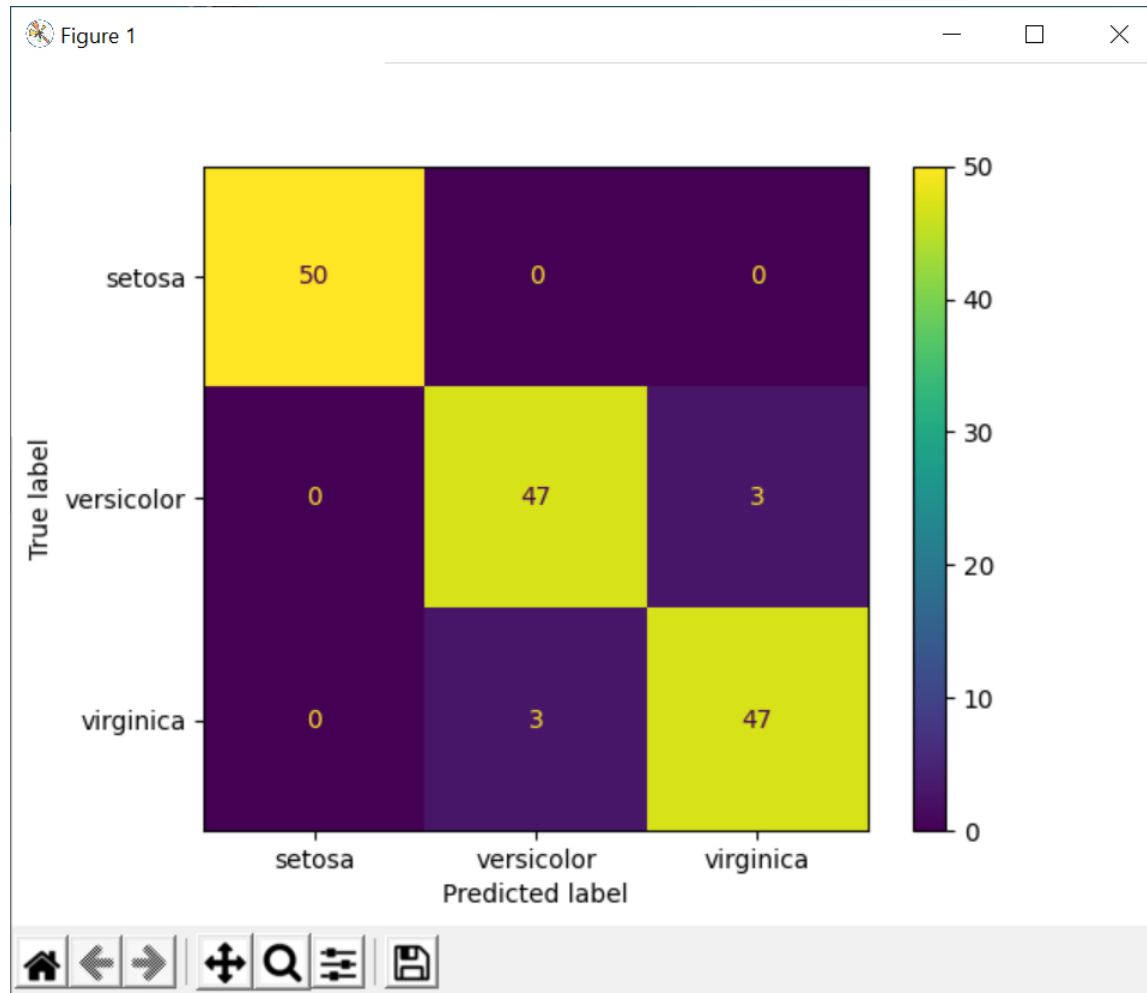
- *matplotlib* is a comprehensive library for visualization in Python

<https://matplotlib.org/>

- Install matplotlib first on the OS command prompt (if not already installed)
> pip install matplotlib

On Python:

```
>>> import matplotlib.pyplot as plt  
>>> disp.plot()  
>>> plt.show()
```



Overall Accuracy = $(50 + 47 + 47) / 150 = 144/150 = 0.96$

Reported Metrics

- **Precision** is the proportion of correct predictions among all predictions of a certain class.

$$Precision = \frac{TP}{TP + FP}$$

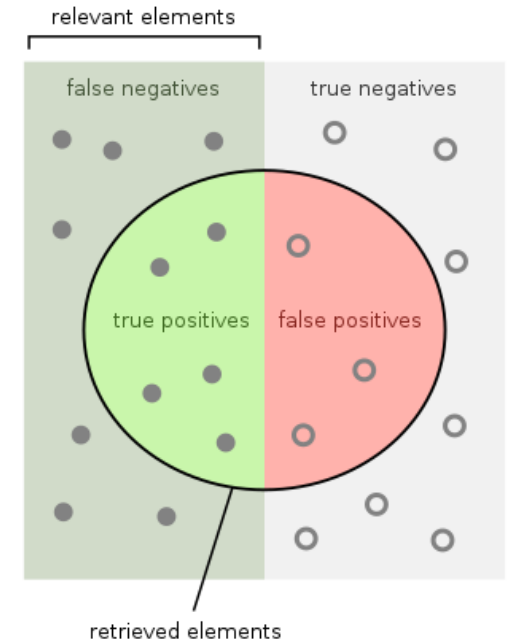
- **Recall** is the proportion of examples of a certain class that have been predicted by the model as belonging to that class. That is, Recall is the proportion of true positives among all true examples.

$$Recall = \frac{TP}{TP + FN}$$

- macro average
 - The unweighted mean per class. This does not take class imbalance into account.
- weighted average
 - The support-weighted mean per class

Quality vs Quantity based on Relevance for Multiple Classes

- **Precision** is the fraction of relevant instances among the retrieved instances.
- **Recall** is the fraction of relevant instances that were retrieved.
- Precision can be seen as a measure of **quality**, and recall as a measure of **quantity**.
- Higher **precision** means that an algorithm returns more relevant results than irrelevant ones.
- higher **recall** means that an algorithm returns more of the relevant results (whether or not irrelevant ones are also returned).



How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

F1 Score

- Sensitivity (Recall) and Precision are to some extent inversely related

$$\text{Sensitivity} = \frac{TP}{P} = \frac{TP}{TP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

- If the number of false positives (FP) increases (meaning that the algorithm is using a broader definition of that class), then the number of false negatives (FN) often decreases, and vice versa.

- F1 score (combing Sensitivity and Precision via a *harmonic mean*)

$$F_1 = 2 \frac{\text{Sensitivity} \times \text{Precision}}{\text{Sensitivity} + \text{Precision}} = \frac{2TP}{2TP + FP + FN} = \frac{TP}{TP + \frac{FP + FN}{2}}$$

Mean of false samples

Classification Report

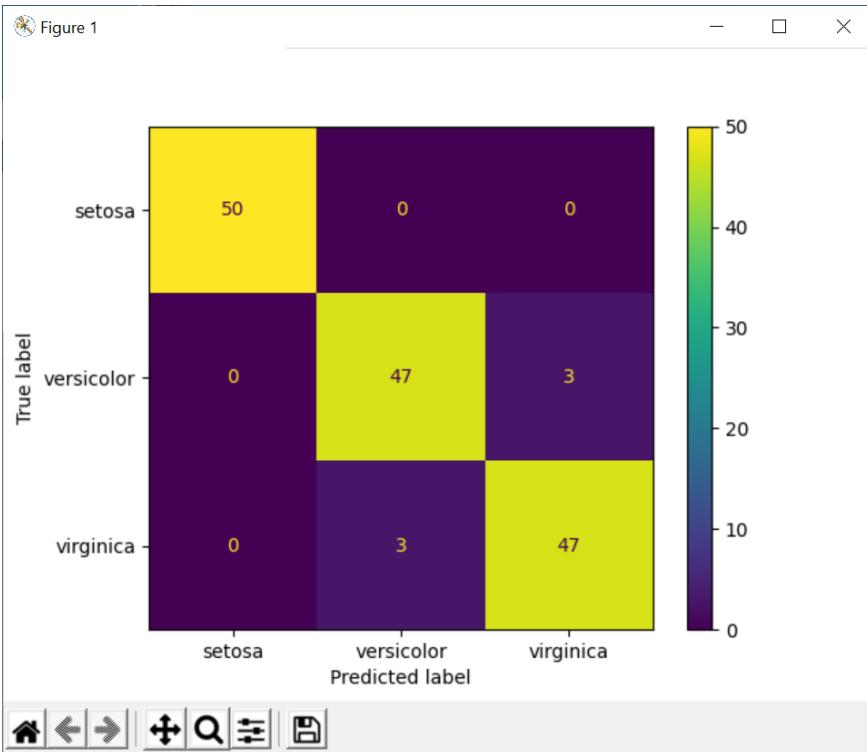
```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(Y, label, target_names =
train_samples.target_names))
```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	50
versicolor	0.94	0.94	0.94	50
virginica	0.94	0.94	0.94	50
accuracy			0.96	150
macro avg	0.96	0.96	0.96	150
weighted avg	0.96	0.96	0.96	150

Note: $0.96 = (1+0.94+0.94)/3$

Conversion to Two Classes

Confusion Matrix for the **relevant** class: versicolor; other classes are **irrelevant**.



True label	Predicted label	
	Versicolor	Others
Versicolor	47 (TP)	3 (FN)
Others	3 (FP)	97 (TN)

$$Precision = \frac{TP}{TP + FP} = \frac{47}{47 + 3} = 0.94$$

$$Recall = \frac{TP}{TP + FN} = \frac{47}{47 + 3} = 0.94$$

$$F_1 = \frac{TP}{TP + \frac{FP + FN}{2}} = 0.94$$